DW_lp_piped_ecc

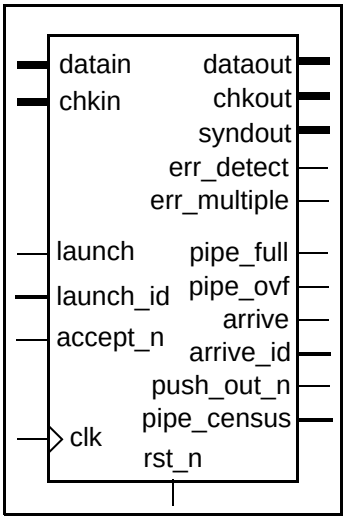
Low Power Pipelined Error Correction Code (ECC)

Version, STAR, and myDesignWare Subscriptions: IP Directory

Features and Benefits

Revision History

- Built-in pipelining and power management
- Automatically enables register retiming
- Operand isolation capability on `datain` and `chkin`
- Parameterized word width
- Parameterized pipeline stages
- Launch identifier tracking propagation
- Generates check bits for written data, corrects single-bit corrupted data for reads, and supports scrubbing
- Automatic single-bit error correction and error syndrome logging
- Error and uncorrectable error indication flags



Description

This component performs pipelined single-error correction, double-error detection (SECDEC) error correction code (ECC) over a parameterized range of word and check bit widths with the option of pipelining.

DW_lp_piped_ecc provides integrated pipeline control that is applied to each pipelined register and the option to configure in operand isolation cells. These features can minimize power consumption and they are only available for DC versions C-2009.06 and later.

The DW_lp_piped_ecc supports pipeline register re-timing, which is automatically enabled for balancing between logic stages, and clock gate insertion capability.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Asynchronous or synchronous reset depending on *rst_mode* parameter (active low)
datain	*data_width* bits	Input	Input data bus
chkin	*chk_width* bits	Input	Input check bits bus (for read or scrubbing)
err_detect	1 bit	Output	Any error flag (active high)

Table 1-1 Pin Description (Continued)

Pin Name	Width	Direction	Function
err_multiple	1 bit	Output	Multiple bit error flag (active high)
dataout	*data_width* bits	Output	Output data bus
chkout	*chk_width* bits	Output	Output check bits bus
syndout	*chk_width* bits	Output	Output error syndrome bus
launch	1 bit	Input	Control to begin a new ECC operation
launch_id	*id_width* bits	Input	Identifier for the corresponding asserted `launch`
pipe_full	1 bit	Output	Upstream notification that pipeline is full
pipe_ovf	1 bit	Output	Status Flag indicating pipeline overflow
accept_n	1 bit	Input	ECC output values accepted from downstream logic (active low)
arrive	1 bit	Output	ECC output results are valid
arrive_id	*id_width* bits	Output	`launch_id` from the originating `launch` that produced the ECC output result
push_out_n	1 bit	Output	Push performed to downstream FIFO element (active low)
pipe_census	M bits	Output	Number of pipeline register levels currently occupied Note: The value of M is equal to the larger of 1 or ceil(log2(*in_reg* + *stages* + *out_reg*)). Example: if *in_reg* = 1, *stages* = 2, *out_reg* = 1, then M = 2.

Table 1-2 Parameter Description

Parameter	Values	Description
data_width	1 to 8178 Default: 8	Bus width of input `datain` and output `dataout`
chk_width	5 to 14 Default: 5	Bus width of input `chkin` and outputs `chkout` and `syndout`
rw_mode	0 or 1 Default: 1	Read or write mode ■ 0: Read mode ■ 1: Write mode

Table 1-2 Parameter Description (Continued)

Parameter	Values	Description
op_iso_mode	0 to 4 Default: 0	Operand isolation mode (controls datapath gating for minPower flow) Allows you to set the style of minPower datapath gating for this module ■ 0: Use the DW_lp_op_iso_mode[a] synthesis variable ■ 1: 'none' ■ 2: 'and' ■ 3: 'or' ■ 4: Preferred gating style: ‘and’ Datapath gating is inserted only when there are no input registers on the operands at the component boundary. When inserted, datapath gating circuits are placed immediately after the input ports of the component (see Figure 1-3 on page 7).
id_width	1 to 1024 Default: 1	Width of `launch_id` and output `arrive_id`
in_reg	0 or 1 Default: 0	Input register control ■ 0: No input register ■ 1: Include input register
stages	1 to 1022 Default: 4	Number of logic stages in the pipeline
out_reg	0 or 1 Default: 0	Output register control ■ 0: No output register ■ 1: Include output register
no_pm	0 or 1 Default: 1	No pipeline management used 0: Use pipeline management 1: Do not use pipeline management
rst_mode	0 or 1 Default: 0	Control of `rst_n` behavior ■ 0: Asynchronous reset ■ 1: Synchronous reset

a. The DW_lp_op_iso_mode synthesis variable is available only in Design Compiler. DW_lp_op_iso_mode sets a global style of datapath gating. To use the global style, set *op_iso_mode* to '0', Note that If the *op_iso_mode* parameter is set to '0' and DW_lp_op_iso_mode is either not set or set to 0', then no datapath gating is inserted for this component.

Table 1-3 Synthesis Implementations

Implementation Name	Implementation	License Feature Required
rtl	Synthesis model	■ DesignWare (P-2019.03 and later) ■ DesignWare-LP[a] (before P-2019.03)

a. For Design Compiler versions before P-2019.03, see "Enabling minPower" on page 20.

Table 1-4 Simulation Models

Model	Function
DW04.DW_LP_PIPED_ECC_CFG_SIM	Design unit name for VHDL simulation
dw/dw04/src/DW_lp_piped_ecc_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_lp_piped_ecc.v	Verilog simulation model source code

Important Note about the *no_pm* Parameter

The setting of the parameter *no_pm* can have an impact on the power savings of the component. Setting *no_pm* to 0 enables pipeline management capability that facilitates maximum power savings when launching activity is not enabled continuously or in long bursts. For more, see "Pipeline Management and Power Savings (no_pm = 0)" on page 10 (*no_pm* = 0). Setting *no_pm* to 1 is better when long continuous bursts of launches or idles are normal (see "Pipeline Management Disabled (no_pm = 1)" on page 10. Figure 1-1 on page 5 and Figure 1-2 on page 6 show the physical distinction between the two settings of *no_pm*.

Block Diagrams

Figure 1-1 shows the DW_lp_piped_ecc configured with *no_pm* = 0.

Figure 1-1 DW_lp_piped_ecc Configured with *no_pm* = 0

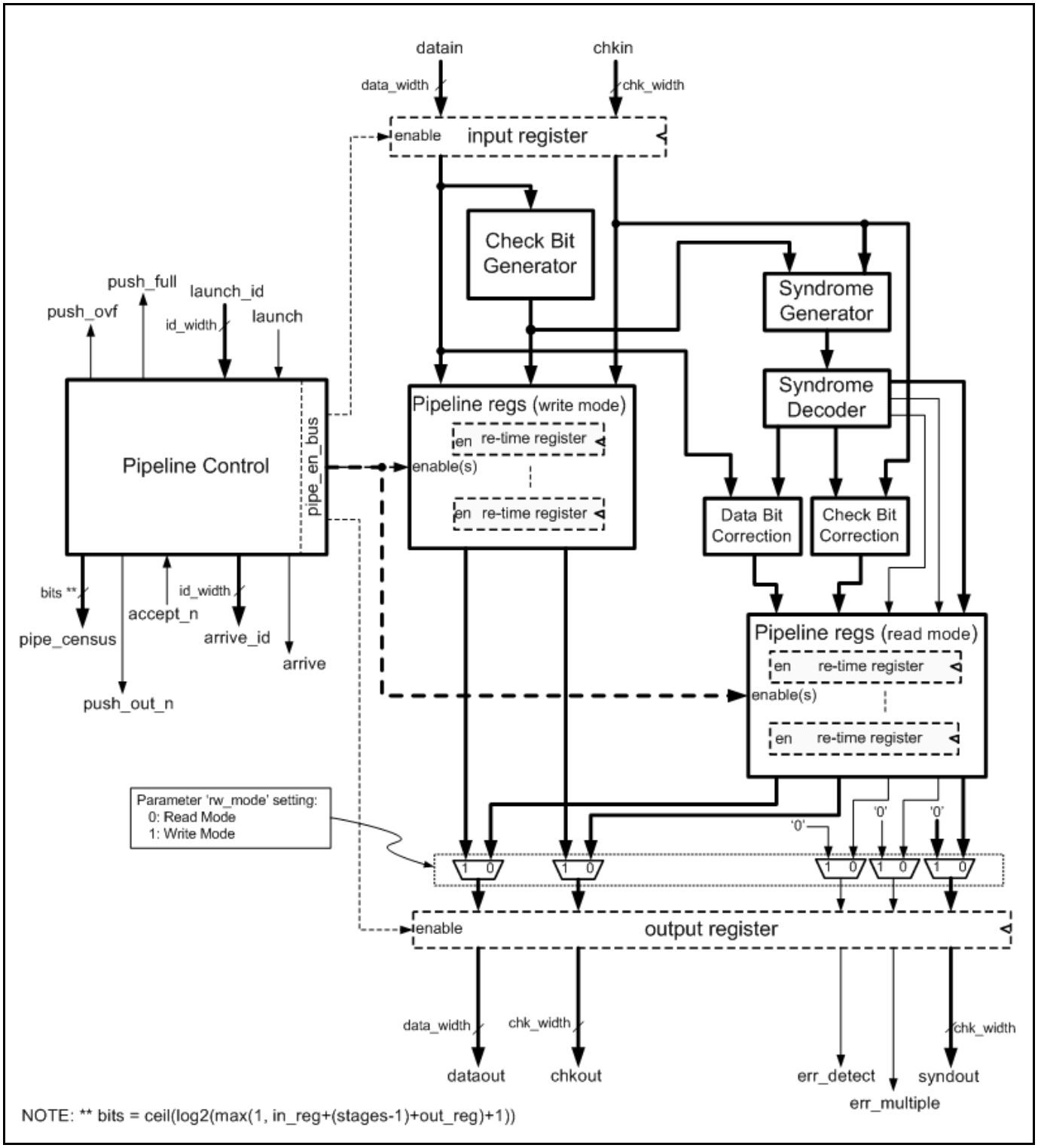


Figure 1-2 shows the DW_lp_piped_ecc configured with *no_pm* = 1.

Figure 1-2 DW_lp_piped_ecc Configured with *no_pm* = 1

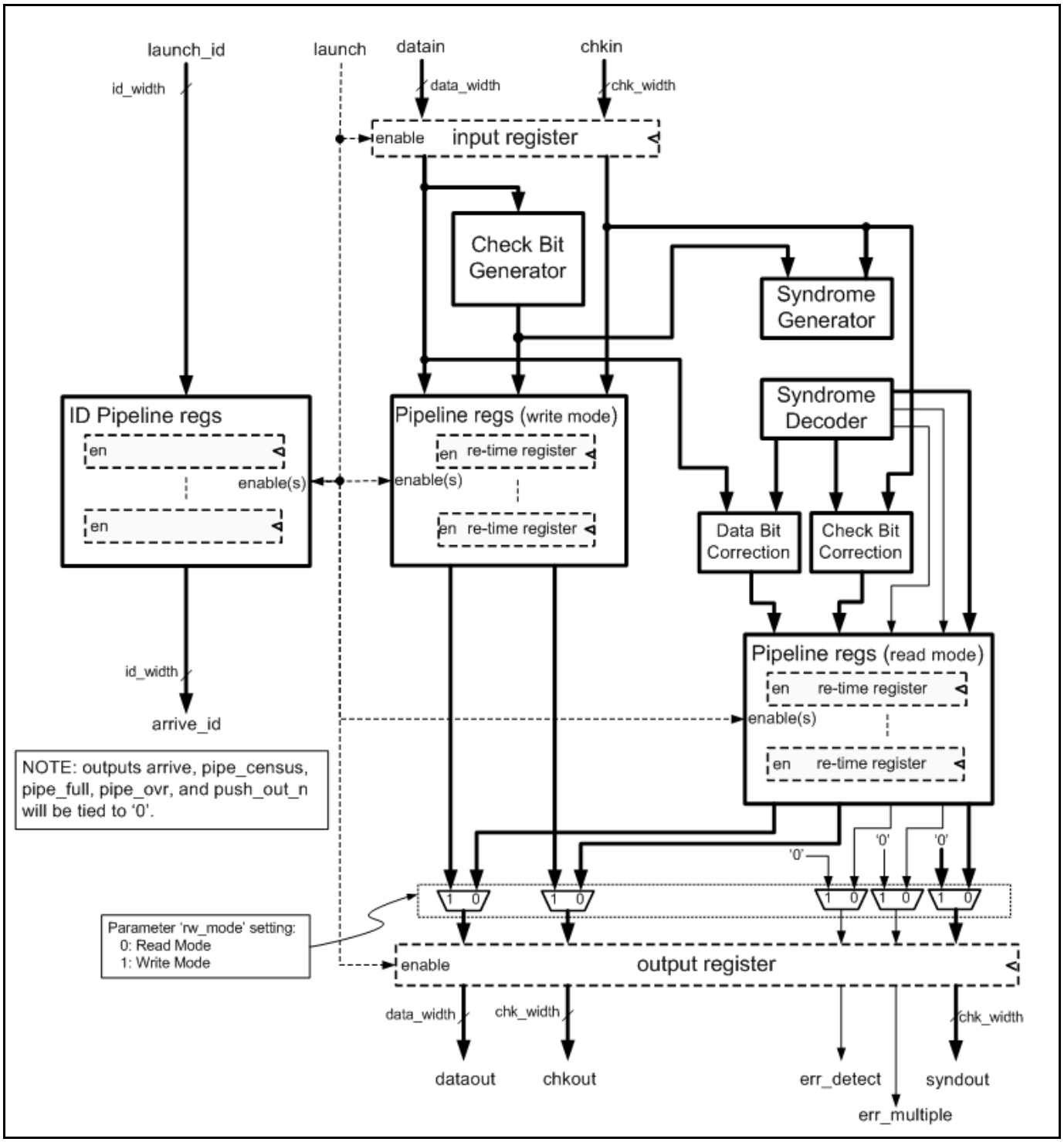
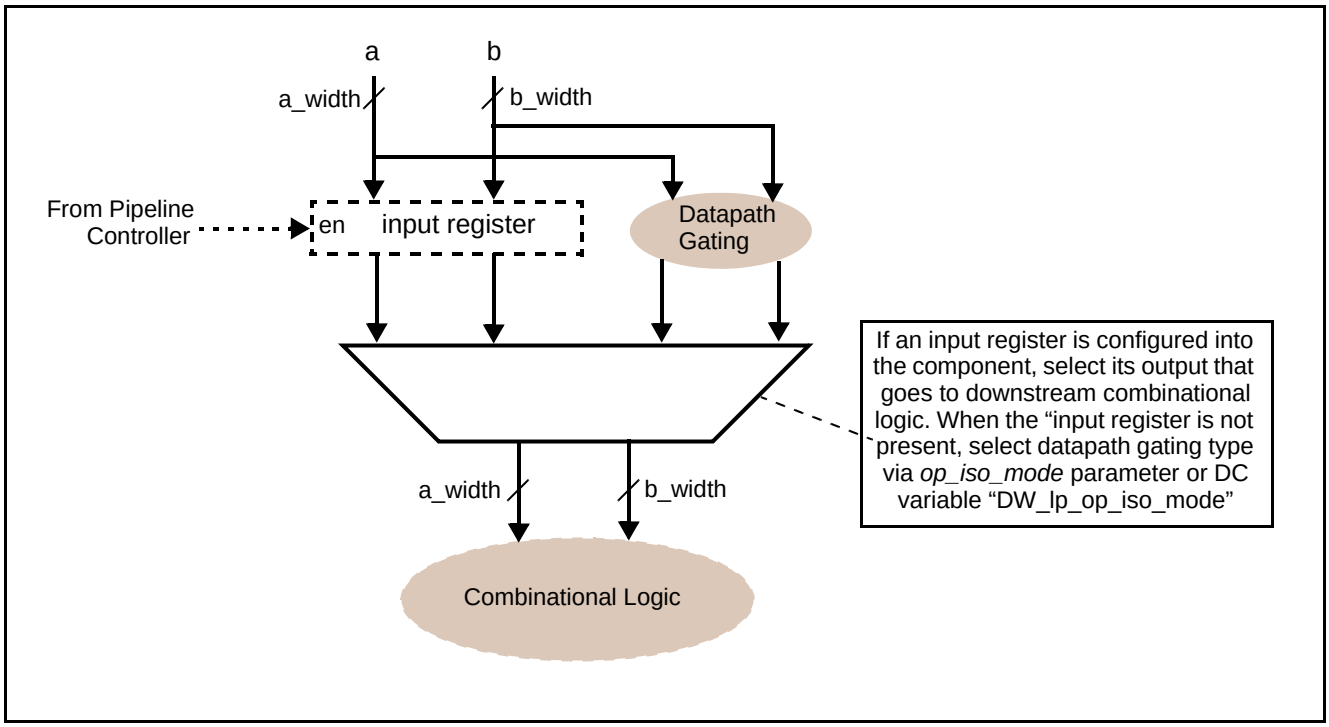


Figure 1-3 shows where datapath gating is inserted when the *op_iso_mode* parameter enables it.

Figure 1-3 Location of Datapath Gating (If Inserted)



Write Mode Operation

The DW_lp_piped_ecc operates in either a write mode or a read mode selected by the parameter *rw_mode*. Write mode (*rw_mode* set to 1) generates check bits from the input data (`datain`). Both the generated check bits (`chkout`) and pass through data (`dataout`) are output.

The check bit encoding is performed by a modified Hamming code (modeled after DW_ecc). When *rw_mode* is 1 and *no_pm* = 0, a new check bit calculation begins based on datain when launch goes to 1. For every clock cycle that launch is 1, a new check bit calculation is initiated with the associate `datain` value. Check bit results along with the associated (unchanged) data are available on the outputs, `chkout` and `dataout`, respectively, when `arrive` is 1. If needed, the `arrive_id` output assists in identifying the results based on the initiated `launch_id` into the pipeline manager (for details, see "Pipeline Management and Power Savings (no_pm = 0)" on page 10).

When *no_pm* = 1, there is no pipeline manager available. The check bit results are pipelined (globally enabled by the `launch` input) with a prescribed amount of clock cycles (while `launch` is1), based on *in_reg*, *stages*, and *out_reg* values that define the number of register level as shown below in Table 1-6 on page 9. To assist in tracking each set of `datain` and `chkin` initiations, transaction identifiers can be driven into `launch_id` and monitored on the pipelined `arrive_id` output to locate the associated results. That is, a unique `launch_id` value for each `datain` will makes its way through the pipeline and show up at `arrive_d` along with the corresponding `dataout` and `chkout` results.

Read Mode Operation

When the *rw_mode* parameter is set to 0 (indicating read mode operation), the check bits are computed from the `datain` inputs and XORed with the `chkin` inputs to form the error syndrome. DW_lp_piped_ecc implements single-error correction and double error detection (SECDEC). Therefore, when a two-bit error is encountered, both `err_detect` and `err_multiple` are high. Single-bit errors, however, are correctable, as indicated by `err_detect` high and `err_multiple` low. If all bits of the syndrome are 0, then there is no error detected (recomputed check bits are the same as `chkin` inputs). If one or more syndrome bits are 1, then an error is detected as indicated by the output `err_detect`=1. If the error is not correctable, then the output `err_multiple` will also be 1. The syndrome output `syndout` indicates which bit is in error, for single-bit errors, and it is useful in systems where it is desirable to keep a log of which bits are chronically in error. Also, `syndout` identifies when a multiple-bit error occurs but without identifying which bits are incorrect.

When `syndout` contains at least one non-zero bit, it indicates an error. Whether it is a single-bit vs. a double-bit error is determined by the number of bits that are 1 in the `syndout`. The modified Hamming code implemented will indicate a single-bit error when `syndout` is non-zero and there is an odd number of 1's and the value of `syndout` correlates to the data bit or check bit in error. Multiple-bit errors are present when `syndout` is non-zero and contains an even number of 1s.

When a correctable error is detected (indicated by `err_detect`=1 and `err_multiple`=0), the input data bit in error is corrected on `datain` and passed to `dataout`. If the single-bit error is in the check bits, then the input check bit that is in error is corrected and `chkin` is passed to `chkout`. In either case where there is a correctable error in data or the check bits, the output syndrome, `syndout`, will be output as non-zero. If the error is not correctable (indicated by `err_multiple`=1), `datain` passes to `dataout` unchanged as well as the check bits remaining unchanged from `chkin` to `chkout`. The `syndout` output will be non-zero for an uncorrectable error, but it will have no meaning as to which bits (in the data or check vectors) were incorrect.

The DW_lp_piped_ecc will always correct a single-bit error.

For *no_pm* = 1, there is no pipeline manager available. The outputs are pipelined results initiated when `launch` is 1 with a prescribed amount of clock cycles (while `launch` is 1) based on *in_reg*, *stages*, and *out_reg* values that define the number of register level as shown in Table 1-6 on page 9. To assist in tracking each set of `datain` and `chkin` initiations, transaction identifiers can be driven into `launch_id` and monitored on the pipelined `arrive_id` output to locate the associated results. That is, a unique `launch_id` value per `datain` and `chkin` pair will makes its way through the pipeline and show up at `arrive_d` along with its corresponding `dataout`, `chkout`, `syndout`, `err_detect`, and `err_multiple` results.

For related ECC usage and application information, see the DW_ecc datasheet.

Parameter Sizing

The number of check bits (*chk_width*) required depends on the width of the data word (represented by *data_width*). Table 1-5 indicates the minimum required *chk_width* setting given the parameter *data_width*. The number of check bits must be passed via the *chk_width* parameter. If a value less than what is stated in Table 1-5 for *chk_width* is set, synthesis will fail to build this component properly.

Table 1-5 Minimum *chk_width* Required for *data_width* Range

data_width	chk_width (min)
1 to 11	5
12 to 26	6
27 to 57	7
58 to 120	8
121 to 247	9
248 to 502	10
503 to 1013	11
1014 to 2036	12
2037 to 4083	13
4084 to 8178	14

Pipelining

The DW_lp_piped_ecc is configurable to embed pipeline register levels. Setting the value for the parameters *in_reg*, *stages*, and *out_reg* (see Table 1-6 on page 9) determines the number of pipeline register levels that are inserted. Therefore, depending on the parameter *in_reg*, *stages*, and *out_reg* settings, the number of clock cycles for the `dataout`, `chkout`, `syndout`, `err_detect`, and `err_multiple` results to propagate varies.

This DW_lp_piped_ecc is designed to make it easy to pipeline the error detection and correction logic using the register retiming features of DC. It also contains parameter controlled input and output registers which will stay in place at their respective block boundary--they are not allowed to be moved by DC register retiming features.

The input and output registers are not available when using DC versions earlier than A-2007.12.

The parameter *stages* refers to the number of logic stages desired after register retiming is performed. The number of register levels is not necessarily the same as the number of logic stages. If no input or output registers are used (*in_reg* = 0 or *out_reg* = 0, respectively), then there is one fewer register level than logic *stages*. If either an input register or output register is specified, then the number of register levels is the same as the number of logic *stages*. If both input and output registers are specified, then the number of register levels is the number of logic *stages* + 1. Table 1-6 summarizes this. The number of pipeline register levels that can be retimed is always *stages* - 1.

Table 1-6 Number of Pipeline Register Levels Based on *in_reg*, *out_reg*, and *stages* Settings

in_reg	out_reg	Number of Pipeline Register Levels
0	0	*stages*-1
0	1	stages
1	0	stages

Table 1-6 Number of Pipeline Register Levels Based on *in_reg*, *out_reg*, and *stages* Settings (Continued)

in_reg	out_reg	Number of Pipeline Register Levels
1	1	*stages*+1

Pipeline Management and Power Savings (*no_pm* = 0)

When the parameter *no_pm* is 0, running in parallel to the pipeline register levels is pipeline control logic (as seen in the block diagram in Figure 1-1 on page 5) that monitors the activity. In cases where there is inactivity on a particular register level of the pipeline, the pipeline control disables those levels to promote power savings. Furthermore, if using the Synopsys Power Compiler tool, the presence of the pipeline control and its wiring to the pipeline register levels provides an opportunity for increased power reduction in the form of clock gating.

The most noticeable power savings compared to setting *no_pm* to 1 occurs when active or inactive launch sequences fall in the range (in number of clock cycles) from 1 to the number of register stages in the pipeline. On average, when launching is absent of long continuous bursts (long bursts -- the number of cycles greater than or equal to the number of pipeline register stages) of either asserted or de-asserted behavior, using *no_pm* = 0 can provide maximum power savings. For example, with a configuration consisting of 4 pipeline register stages, an enabled launch sequence that is active every other clock cycle will produce lower power usage than a launch sequence that is asserted for 5 consecutive cycles then de-asserted for 5 consecutive cycles.

Along with the potential power savings that the pipeline control provides, it can improve performance when intermittent launch operations are present and there are first-in-first-out (FIFO) structures upstream and downstream of the DW_lp_piped_ecc. The handshake between the DW_lp_piped_ecc and the external FIFOs is via the `accept_n` and `pipe_full` ports. Effectively, the DW_lp_piped_ecc can be considered part of the external FIFO structures. The performance gain comes when inactive (bubbles) stages are detected. These pipeline 'bubbles' are removed to produce a contiguous set of active pipeline stages. The result is empty pipeline slots at the head of (or entering) the DW_lp_piped_ecc pipeline for new operations to be launched. Advancing the shifting of operations through the pipeline when a valid product result is available (`arrive` = 1) is controlled by the `accept_n` input. When the pipeline is full of active entries, the `pipe_full` output is 1. To disable this feature in cases where no external FIFOs are present, set the `accept_n` input to 0 which will effectively eliminate any flow control. At the same time, the `pipe_full` output would always be 0.

To assist in tracking of 'launched' operands, the pipeline control logic provides interface ports called `launch_id` and `arrive_id`. The `launch_id` input is assigned a value during an active launch operation. Given that `launch_id` values are unique in successive launch operations, the `dataout`, `chkout`, `syndout`, `err_detect`, and `err_multiple` results can be distinguished from one another with the assertion of arrive and the associated `arrive_id`. The `arrive_id` is the `launch_id` from the originating launch that produced the valid output results.

Pipeline Management Disabled (*no_pm* = 1)

When the *no_pm* is 1, a pipeline that is enabled by the `launch` input as shown in Figure 1-2 on page 6 which implies that there is no pipeline control logic. Individual pipeline enables are not available in this configuration, but all register enables are tied together and connected to the `launch` input. So, the entire pipeline is always enabled when `launch` is 1 or disabled when `launch` is 0. This still allows for Synopsys

Design Compiler to insert clock gating how it deems necessary. But the pipeline "bubble removal" characteristics described when *no_pm*=0 (immediately above) is nonexistent. However, the register retiming feature is automatically enabled as well as the capability of clock gate insertion. Even though pipeline control is not available, the feature of providing a tracking identification number is maintained via the `launch_id` and `arrive_id`. All other inputs and outputs related to pipeline control when *no_pm*=0 are either not used (for inputs) or driven to logic 0 for the outputs when *no_pm* = 1. Table 1-7 summarizes the inputs not used and outputs driven to logic 0 when DW_lp_piped_ecc is configured with *no_pm* = 1. Figure 1-8 on page 18 shows the behavior of this configuration.

If launching behavior contains long continuous bursts (number of cycles greater than or equal to pipeline register stages), either active or non-active, setting *no_pm* to 1 is recommended to minimize power consumption and area. In these cases, bubble removal opportunities will not occur frequently enough to warrant using pipeline control (*no_pm* = 0) to optimize power benefit as described in the previous section.

Table 1-7 Pipeline Control Pins Function when *no_pm* = 1

Pin Name	Width	Direction	Function
launch	1 bit	Input	Globally connects to all pipeline register enables
pipe_full	1 bit	Output	Fixed to 0
pipe_ovf	1 bit	Output	Fixed to 0
accept_n	1 bit	Input	Unconnected internally
arrive	1 bit	Output	Fixed to 0
push_out_n	1 bit	Output	Fixed to 0
pipe_census	M bits	Output	Fixed to all 0 Note: The value of M is equal to the larger of 1 or ceil(log2(*in_reg*+*stages*+*out_reg*)). Example: if *in_reg*=1, *stages*=2, *out_reg*=1, then M=2.

No Pipeline Register Levels Specified

In cases where no pipelining is required through the DW_lp_piped_ecc (*in_reg* = 0, *stages* = 1, and *out_reg* = 0) and *no_pm*=0, the pipeline control flow control handshaking/status signals still remain active and meaningful with one exception. The `pipe_census`, which is intended to count the number of active pipeline register levels, becomes irrelevant and is fixed to 0. For waveforms, see Figure 1-7 on page 17.

System Resets (Synchronous or Asynchronous)

Two system reset modes are available from the `rst_n` input: asynchronous or synchronous. Asynchronous system reset is implemented when *rst_mode* is 0 and synchronous system reset is applied when *rst_mode* is 1. Figure 1-9 on page 19 and Figure 1-10 on page 20 depict the behaviors of asynchronous and synchronous resets, respectively.

During reset conditions, all the output ports are set to 0.

Initializing Test Bench Memories With ECC Check Bits

The Verilog function, DWF_ecc_gen_chkbits, is provided in the file, $SYNOPSYS/dw/sim_ver/DW_ecc_function.inc. This function can be used in initial blocks to calculate check bits for the initialization of memories before simulation begins. To allow the function to be used with multiple configurations of DW_ecc, it accepts the data width and check bit width values as arguments along with the data to calculate check bits on and always returns a 16-bit result. For example, if the three check bits are to be calculated on an eight bit data value in the variable my_data, the function call might look like this:

```
`include "DW_ecc_function.inc"
  initial begin : initialize_stuff
    reg [15:0] dummy_chkbits;
    dummy_chkbits = DWF_ecc_gen_chkbits(8, 5, my_data);
    my_check_bits  = dummy_chkbits[4:0];
  end // initialize_stuff
```

This assumes that the variable my_check_bits is declared as a `reg` variable of size five bits and that the directory $SYNOPSYS/dw/sim_ver has been added to the simulator's include search path.

The DWF_ecc_gen_chkbits function is configured, by default, to support data widths up to 2048 bits. This can be changed by defining the Verilog macro, DW_ecc_func_max_width, up to the maximum allowed by the DW_ecc component.

The DWF_ecc_gen_chkbits function is provided for simulation only and is not capable of synthesis.

The DW_ecc_function.inc file contains a comment header that has additional information including the function argument descriptions. See “Header Information for DW_ecc_function.inc” on page 25 for the contents of this header.

Suppressing Warning Messages During Verilog Simulation

The Verilog simulation model includes macros that allow you to suppress warning messages during simulation.

To suppress all warning messages for all DWBB components, define the DW_SUPPRESS_WARN macro in either of the following ways:

- Specify the Verilog preprocessing macro in Verilog code:

  ```
  `define DW_SUPPRESS_WARN
  ```

- Or, include a command line option to the simulator, such as:

  `+define+DW_SUPPRESS_WARN` (which is used for the Synopsys VCS simulator)

The warning messages for this model include the following:

- If values other than 1 or 0 are present on a clock port, the following message is displayed:

  ```
  WARNING: <instance_path>.<clock_name>_monitor:
      at time = <timestamp>, Detected unknown value, x, on <clock_name> input.
  ```

  To suppress only this warning message for all DWBB components, use the following macro:

  - Define the DW_DISABLE_CLK_MONITOR macro. You can define this macro in the following ways:
    - Specify the Verilog preprocessing macro in Verilog code:

      ```
      `define DW_DISABLE_CLK_MONITOR
      ```

    - Or, include a command line option to the simulator, such as:

      `+define+DW_DISABLE_CLK_MONITOR` (which is used for the Synopsys VCS simulator)

  This message is also suppressed using the DW_SUPPRESS_WARN macro explained earlier.

Timing Waveforms

Figure 1-4 on page 14 shows a case where there are two pipeline register levels since *in_reg* is 1, *stages* is 1, and *out_reg* is 1. Launching is performed while `accept_n` is de-asserted causing the pipeline to fill up. This is indicated by `pipe_full` going to 1 while `accept_n` is 1. The `pipe_census[1:0]` value is 2 which indicates that all the pipeline register levels contain active results. Notice that the first `dataout[15:0]` result of 'f9c6' is available as indicated by `arrive` being 1. Also, the identification tracking from the `launch_id[3:0]` of 1 is seen upon arrival with the matching value on `arrive_id[3:0]`.

At the point that the pipeline is full, `accept_n` is asserted (0) to begin emptying the pipeline. Note that `pipe_full` de-asserts when `accept_n` is asserted, but the `pipe_census[1:0]` value still indicates 2 the next clock cycle since a launch coincided with the asserted `accept_n`. Once the launching activity ceases, the continued assertion of `accept_n` drains the pipeline of active results with `pipe_census[1:0]` eventually going to 0.

Figure 1-4 Launching Until Full, Accepting Until Empty

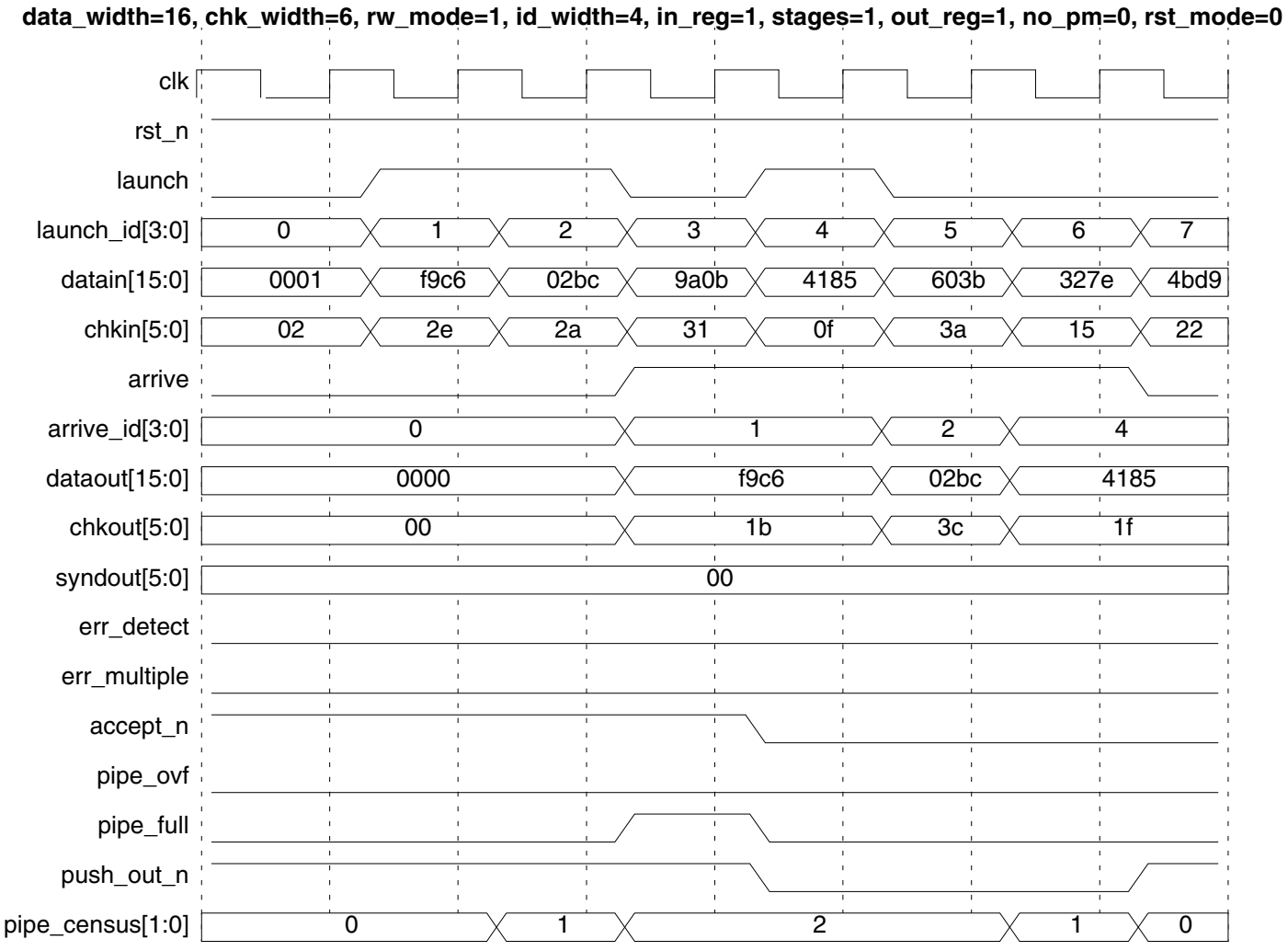


Figure 1-5 shows a case where `launch` is asserted every other clock cycle while `accept_n` is always asserted (0). There are two pipeline register levels because *in_reg* is 1, *stages* is 1, and *out_reg* is 1. So, the first `dataout[15:0]` result of 2280 and `chkout[5:0]` of 19 (`arrive_id[3:0]` of 1) arrive after the second rising-edge of `clk` from the first asserted `launch` with accompanying `launch_id[3:0]` of 1. Any values of `datain[15:0]` are ignored when `launch` is 0.

While in write mode (*rw_mode* = 1), any values of `chkin` are disregarded in any event and a new `chkout` is calculated from each `datain` value.

Figure 1-5 Launch Every Other Cycle with accept_n Asserted

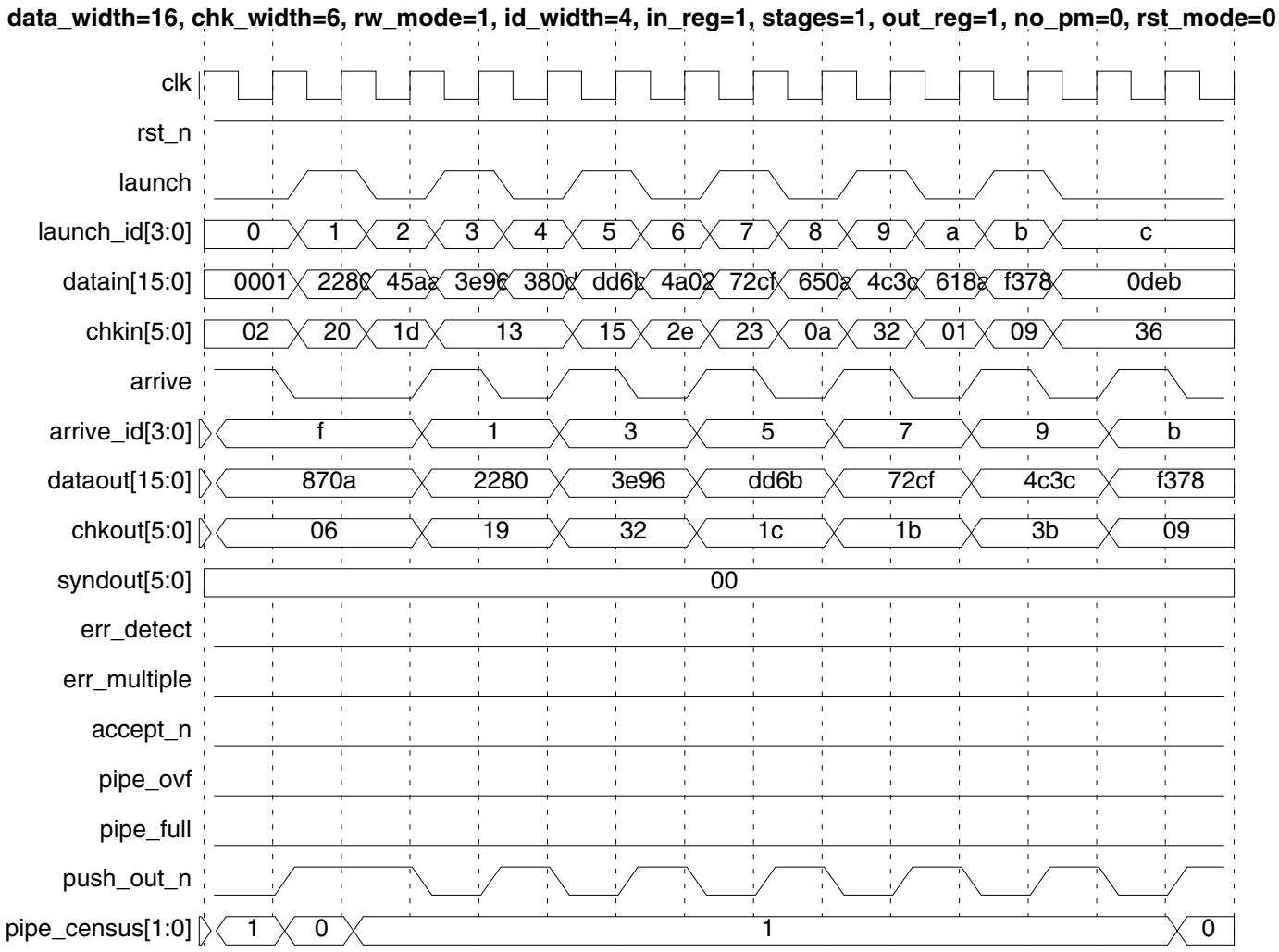


Figure 1-6 depicts a pipeline overflow condition. This is configured as "read mode" (*rw_mode* is 0) and the number of pipeline register levels is 2 (*in_reg* is 0, *stages* is 2, and *out_reg* is 1). The `pipe_ovf` output is registered and gets asserted following the rising-edge of `clk` when the pipeline is full (`pipe_full` is 1), `launch` is asserted (1), and `accept_n` is not asserted (1). In this situation, the launched operation is ignored and the pipeline contents are preserved. `pipe_ovf` does not de-assert until an asserted `accept_n` is sampled by rising-edge of `clk`.

Figure 1-6 Pipeline Overflow

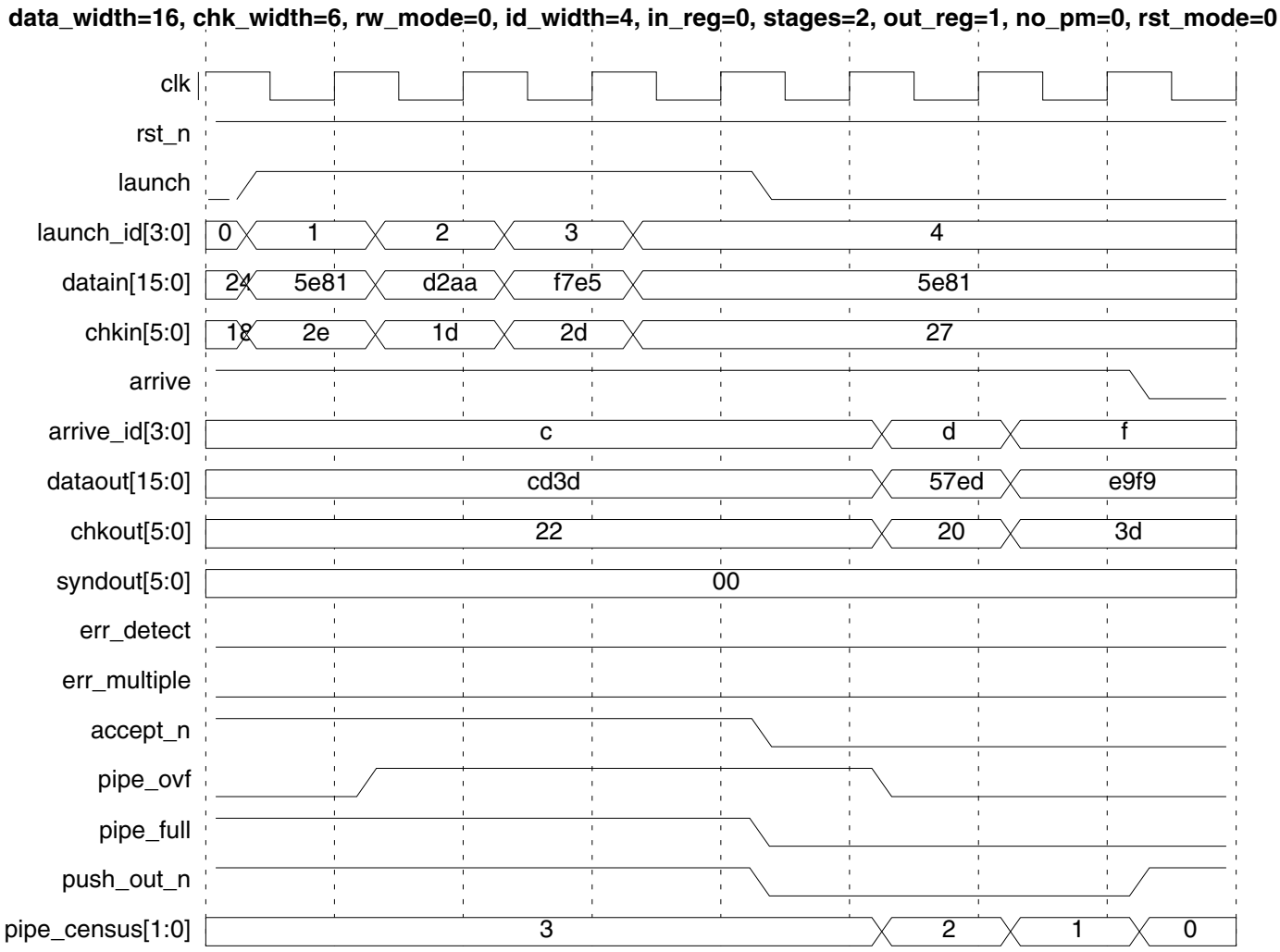


Figure 1-7 depicts a scenario when there is no pipelining configured into the DW_lp_piped_ecc which is defined when *in_reg* is 0, *stages* is 1, and *out_reg* is 0, but *no_pm* is 0. Thus, in this case where write mode is enabled (*rw_mode* = 1), the `dataout[7:0]` are `chkout[4:0]` outputs are a pure combinational logic path from `datain[7:0]`. The flow control/status outputs `arrive`, `arrive_id[7:0]`,`pipe_full`,`pipe_ovf`,`push_out_n` still have meaning. However, the output `pipe_census` has no meaning since no pipeline register levels exist. Hence, `pipe_census` will always be driven to 0.

Note This configuration is different from a case when *no_pm* = 1 (see Figure 1-8 on page 18), where here most of the pipeline controller outputs are meaningful. When *no_pm* is 1, the same pipeline controller outputs are meaningless except for `arrive_id`.

Figure 1-7 No Pipeline Case (*in_reg* = 0, *stages* = 1, *out_reg* = 0, *no_pm* = 0)

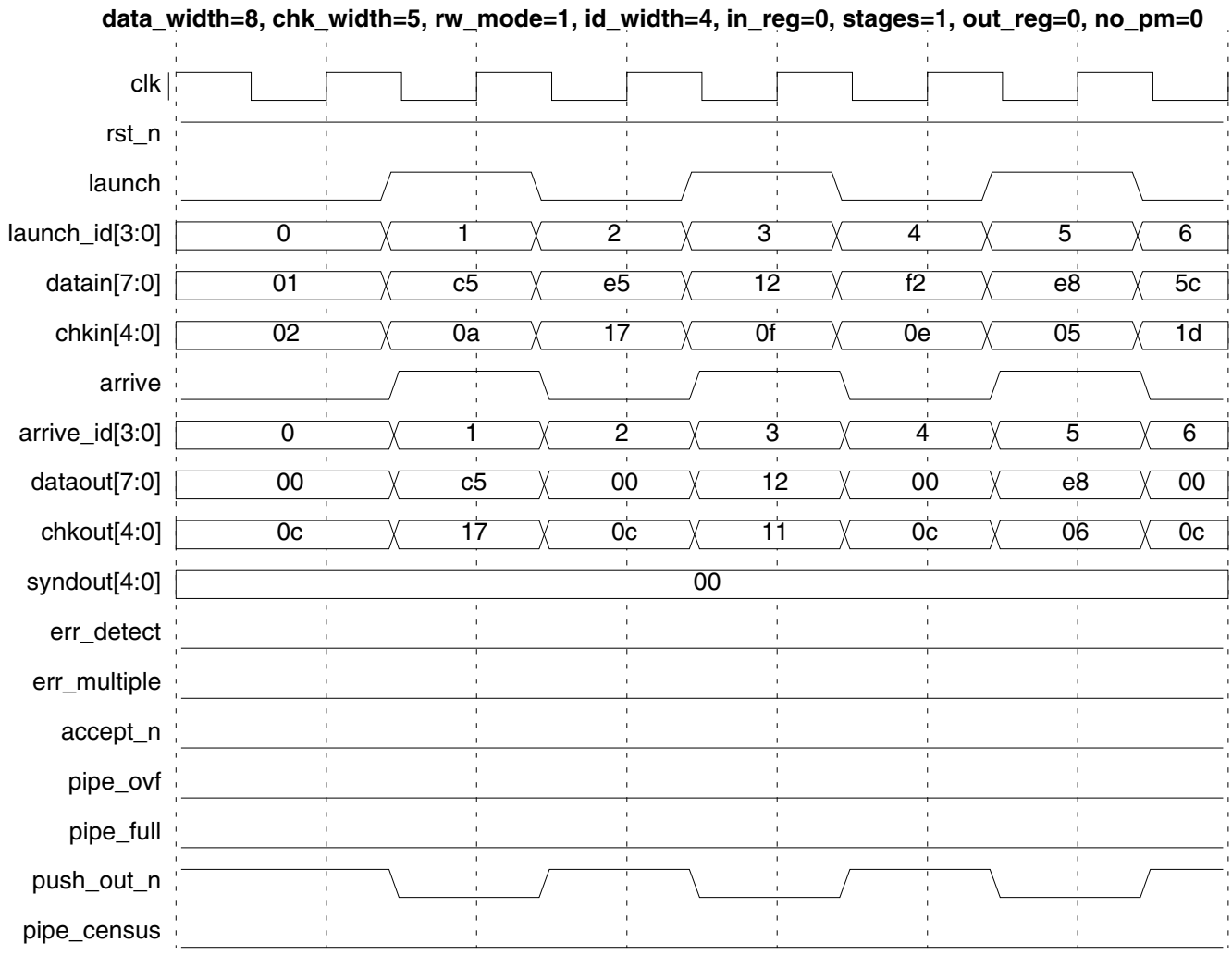


Figure 1-8 is a case where the pipeline controller is not in existence with *no_pm* set to 1. Therefore, the pipeline register levels in this case, 2, are enabled only when `launch` is 1. The only portion of the pipeline controller interface this is maintained is the identification tracking logic driven by `launch_id[3:0]` that produces `arrive_id[3:0]`. The tracking identifier works just like the case when *no_pm* is 0 except the flow control capabilities are non-existent under this configuration.

Notice that when `arrive_id[3:0]` is 5, the `err_detect` is 1 and the `err_multiple` is 0. This indicates that a single-bit error was detected and corrected. The `syndout[5:0]` value being non-zero (16) implies this and either `dataout` will be corrected from `datain` or `chkout` corrected from `chkin`. In this case, `datain` was in error as seen when `launch_id[3:0]` is 5, `datain` was 5e91. The corresponding `dataout` was corrected to be 5e81 which indicates that bit 1 of `datain` was in error.

Figure 1-8 No Pipeline Controller (*no_pm* = 1)

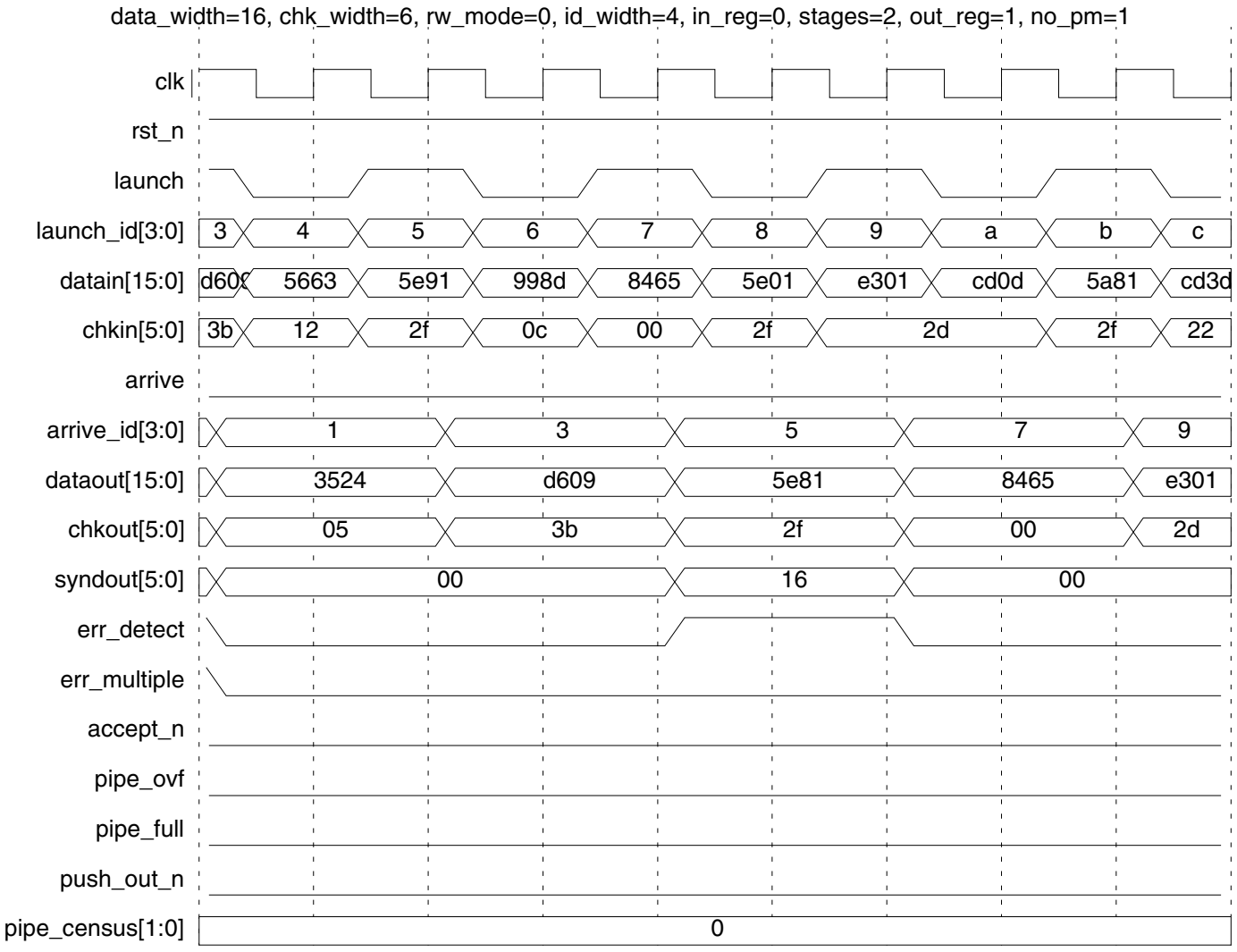


Figure 1-9 shows the affects that the assertion of `rst_n` while configured for asynchronous resetting (*rst_mode* = 0).

Figure 1-9 Asynchronous Reset Behavior (*rst_mode* = 0)

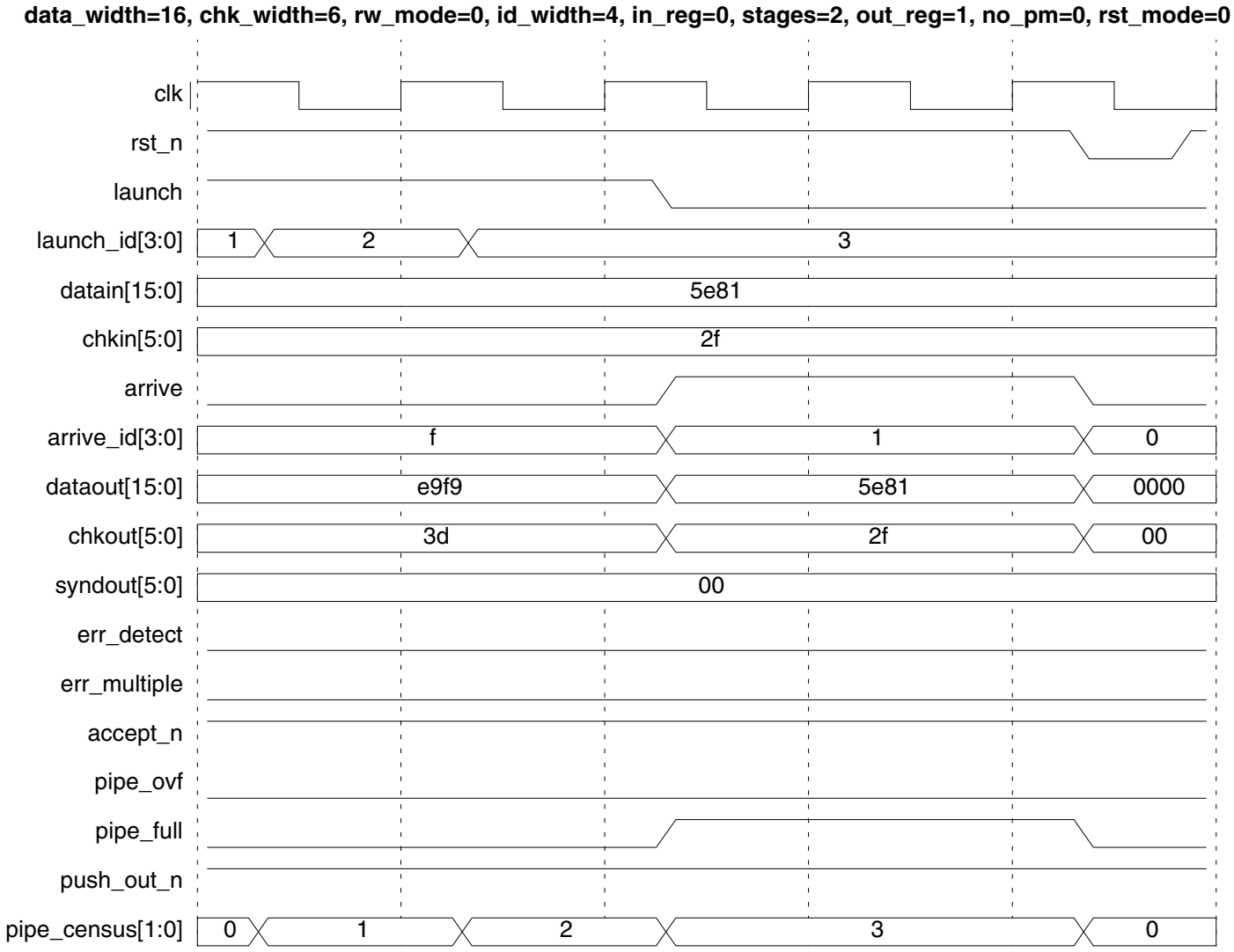
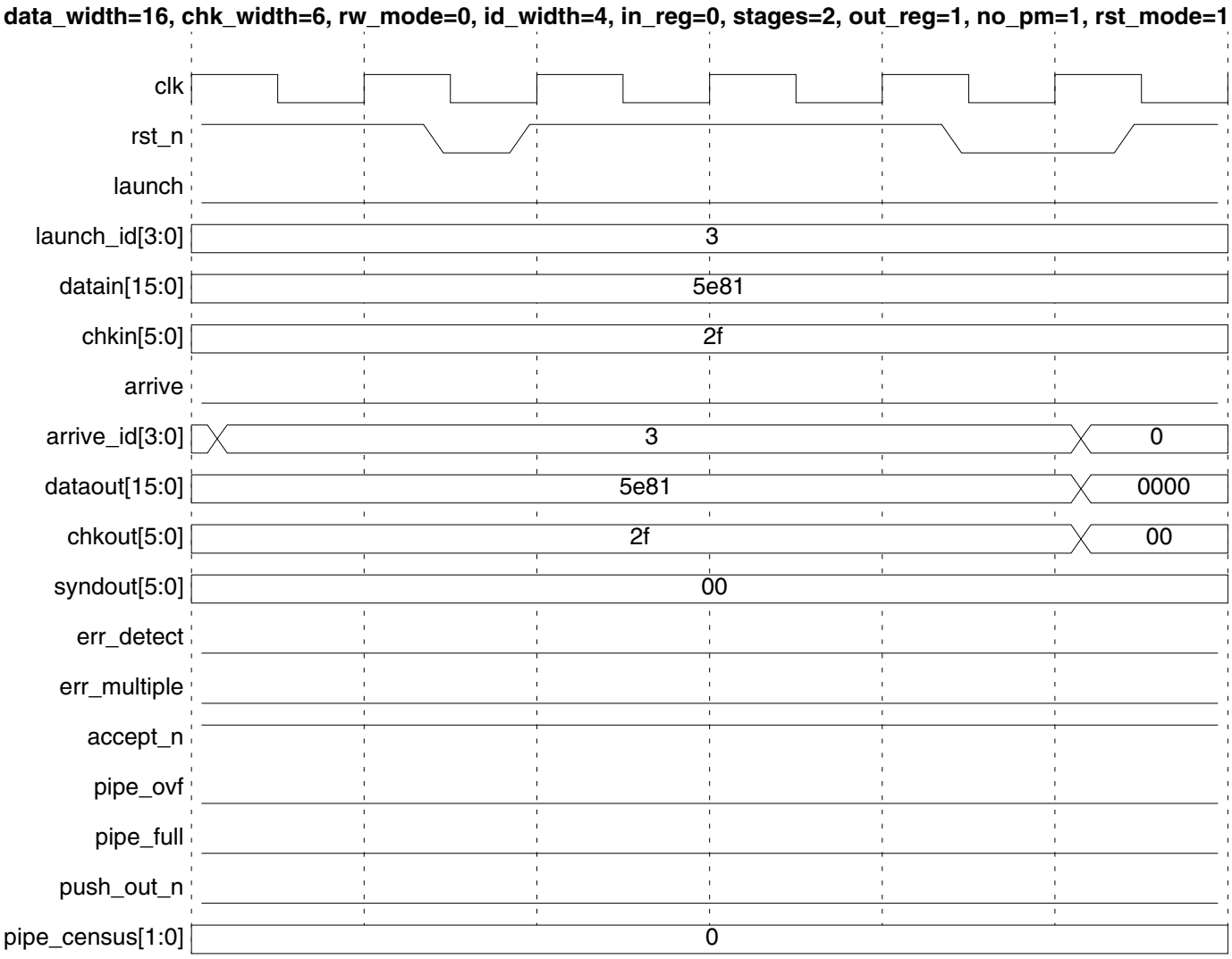


Figure 1-10 shows the affects of asserting `rst_n` while configured for synchronous resetting (*rst_mode* = 1).

Register elements are cleared only when `rst_n` = 0 is sampled at the rising-edge of `clk`.

Figure 1-10 Synchronous Reset Behavior (*rst_mode* = 1)



Enabling minPower

In Design Compiler (version P-2019.03 and later) and Fusion Compiler, you can instantiate this component and use all its features without special settings.

For versions of Design Compiler before P-2019.03, enable minPower as follows:

```
set synthetic_library {dw_foundation.sldb dw_minpower.sldb}
set link_library {* $target_library $synthetic_library}
```

Related Topics

- DesignWare Building Blocks User Guide

HDL Usage Through Component Instantiation - VHDL

```
library IEEE,DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DWpackages.all;
use DWARE.DW_Foundation_comp.all;

entity DW_lp_piped_ecc_inst is
      generic (
         inst_data_width : POSITIVE := 16;
         inst_chk_width : POSITIVE := 6;
             inst_rw_mode : NATURAL  := 1;
             inst_op_iso_mode : NATURAL  := 0;
             inst_id_width : POSITIVE := 1;
             inst_in_reg : NATURAL  := 0;
             inst_stages : POSITIVE := 3;
             inst_out_reg : NATURAL  := 0;
             inst_no_pm : NATURAL  := 1;
             inst_rst_mode : NATURAL  := 0
         );
       port (
             inst_clk : in std_logic;
             inst_rst_n : in std_logic;
             inst_datain : in std_logic_vector(inst_data_width-1 downto 0);
             inst_chkin : in std_logic_vector(inst_chk_width-1 downto 0);
             err_detect_inst : out std_logic;
             err_multiple_inst : out std_logic;
             dataout_inst : out std_logic_vector(inst_data_width-1 downto 0);
             chkout_inst : out std_logic_vector(inst_chk_width-1 downto 0);
             syndout_inst : out std_logic_vector(inst_chk_width-1 downto 0);
             inst_launch : in std_logic;
             inst_launch_id : in std_logic_vector(inst_id_width-1 downto 0);
             pipe_full_inst : out std_logic;
             pipe_ovf_inst : out std_logic;
             inst_accept_n : in std_logic;
             arrive_inst : out std_logic;
             arrive_id_inst : out std_logic_vector(inst_id_width-1 downto 0);
             push_out_n_inst : out std_logic;
             pipe_census_inst : out std_logic_vector(1 downto 0)
         );
    end DW_lp_piped_ecc_inst;


architecture inst of DW_lp_piped_ecc_inst is

begin

    -- Instance of DW_lp_piped_ecc
    U1 : DW_lp_piped_ecc
     generic map  ( data_width => inst_data_width,
                    chk_width => inst_chk_width,
                       rw_mode => inst_rw_mode,
                    op_iso_mode => inst_op_iso_mode,
```

```
                    id_width => inst_id_width,
                    in_reg => inst_in_reg,
                    stages => inst_stages,
                    out_reg => inst_out_reg,
                    no_pm => inst_no_pm,
                    rst_mode => inst_rst_mode )

    port map ( clk => inst_clk,
                   rst_n => inst_rst_n,
                   datain => inst_datain,
                   chkin => inst_chkin,
                   err_detect => err_detect_inst,
                   err_multiple => err_multiple_inst,
                   dataout => dataout_inst,
                   chkout => chkout_inst,
                   syndout => syndout_inst,
                   launch => inst_launch,
                   launch_id => inst_launch_id,
                   pipe_full => pipe_full_inst,
                   pipe_ovf => pipe_ovf_inst,
                   accept_n => inst_accept_n,
                   arrive => arrive_inst,
                   arrive_id => arrive_id_inst,
                   push_out_n => push_out_n_inst,
                   pipe_census => pipe_census_inst );
end inst;

-- Configuration for use with a VHDL simulator
-- pragma translate_off
library DW04;
configuration DW_lp_piped_ecc_inst_cfg_inst of DW_lp_piped_ecc_inst is
  for inst
  end for; -- inst
end DW_lp_piped_ecc_inst_cfg_inst;
-- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_lp_piped_ecc_inst( inst_clk, inst_rst_n, inst_datain, inst_chkin,
    err_detect_inst, err_multiple_inst, dataout_inst, chkout_inst, syndout_inst,
    inst_launch, inst_launch_id, inst_accept_n, arrive_inst, arrive_id_inst,
    pipe_full_inst, pipe_ovf_inst, push_out_n_inst, pipe_census_inst );

parameter data_width = 27;
parameter chk_width = 7;
parameter rw_mode = 0;
parameter op_iso_mode = 0;
parameter id_width = 1;
parameter in_reg = 0;
parameter stages = 2;
parameter out_reg = 1;
parameter no_pm = 1;
parameter rst_mode = 0;


`define census_width 2  // ceil(log2(max(1, in_reg+(stages-1)+out_reg)+1))

input inst_clk;
input inst_rst_n;
input [data_width-1:0] inst_datain;
input [chk_width-1:0] inst_chkin;
output err_detect_inst;
output err_multiple_inst;
output [data_width-1:0] dataout_inst;
output [chk_width-1:0] chkout_inst;
output [chk_width-1:0] syndout_inst;
input inst_launch;
input [id_width-1 : 0] inst_launch_id;
input inst_accept_n;
output arrive_inst;
output [id_width-1 : 0] arrive_id_inst;
output pipe_full_inst;
output pipe_ovf_inst;
output push_out_n_inst;
output [`census_width-1 : 0] pipe_census_inst;

    // Instance of DW_lp_piped_ecc
    DW_lp_piped_ecc #(data_width, chk_width, rw_mode, op_iso_mode, id_width, in_reg,
      stages, out_reg, no_pm, rst_mode)
       U1 ( .clk(inst_clk),
                .rst_n(inst_rst_n),
                .datain(inst_datain),
                .chkin(inst_chkin),
                .err_detect(err_detect_inst),
                .err_multiple(err_multiple_inst),
                .dataout(dataout_inst),
                .chkout(chkout_inst),
                .syndout(syndout_inst),
                .launch(inst_launch),
```

```
                .launch_id(inst_launch_id),
                .accept_n(inst_accept_n),
                .arrive(arrive_inst),
                .arrive_id(arrive_id_inst),
                .pipe_full(pipe_full_inst),
                .pipe_ovf(pipe_ovf_inst),
                .push_out_n(push_out_n_inst),
                .pipe_census(pipe_census_inst)
                );

endmodule
```

Header Information for DW_ecc_function.inc

```
////////////////////////////////////////////////////////////////////////////////
//
// NOTE: THIS FUNCTION IS NOT FOR SYNTHESIS. It is provided for use in
//       initializing ECC check bits for memories at time zero of simu-
//       lations (such as in an initial block with 'for' loops).
//
//
// Function: DWF_ecc_gen_chkbits
//
//     Calculates the check bits to be used with the data passed (based on
//     the data width and check bit width arguments passed) that are directly
//     compatible with the logic of the DW_ecc design with parameter values
//     that match the same data and check bit width values.
//
// Arguments:
//     width : number of data bits to calculate check bits from (integer)
//     chkbits : number of check bits to be generated (integer)
//     data_in : data bits to be used to calculate check bits from (vector
//               of size `DW_ecc_func_max_width)
//
// Returns:
//     16-bit value with the specified number of check bits in the low
//     order bits.
//
//
// The Verilog macro, DW_ecc_func_max_width should be set to a value that
// is equal to or greater than the largest value of the width argument passed
// to DWF_ecc_gen_chkbits.  Its default setting (if not defined before the
// DWF_ecc_function.inc file is included) is 2048 but the DW_ecc design can
// support widths up to 8178.  This macro can also be used in the design
// that includes it to size the vector that is passed as the data_in argument
// of the function call (to avoid lint issues).
//
// The DW_ecc_gen_chkbits function always returns a 16-bit value, no matter
// how many check bits are specified with the chkbits argument.
//
////////////////////////////////////////////////////////////////////////////////
```

Revision History

For notes about this release, see the *DesignWare Building Block IP Release Notes*.

For lists of both known and fixed issues for this component, refer to the STAR report.

For a version of this datasheet with visible change bars, click here.

Date	Release	Updates
July 2023	DWBB_202212.5	■ Updated version and date
July 2020	DWBB_201912.5	■ Adjusted content and title of "Suppressing Warning Messages During Verilog Simulation" on page 13 and added the DW_SUPPRESS_WARN macro
October 2019	DWBB_201903.5	■ Added the "Disabling Clock Monitor Messages" section
March 2019	DWBB_201903.0	■ Clarified the op_iso_mode parameter in Table 1-2 on page 2 ■ Clarified licensing requirements in Table 1-3 on page 4 ■ Added Figure 1-3 on page 7 to clarify datapath gating ■ Replaced a reference to an out-dated application note with link to the DW_ecc datasheet ■ Added "Enabling minPower" on page 20 ■ Added this Revision History table and the document links on this page

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.

All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
www.synopsys.com